Spectator: Detection and Containment of JavaScript Worms
Web Application Security Arena

- Web application vulnerabilities are everywhere

- Cross-site scripting (XSS)
  - Dominates the charts
  - “Buffer overruns of this decade”
  - Key enabler of JavaScript worms
The Samy Worm

- Unleashed by Samy as a proof-of-concept in October 2005

<table>
<thead>
<tr>
<th>Worm name</th>
<th>Type of site</th>
<th>Release date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samy/MySpace</td>
<td>Social networking</td>
<td>Oct-05</td>
</tr>
<tr>
<td>xanga.com</td>
<td>Social networking</td>
<td>Dec-05</td>
</tr>
<tr>
<td>SpaceFlash/MySpace</td>
<td>Social networking</td>
<td>Jul-06</td>
</tr>
<tr>
<td>Yamanner/Yahoo! Mail</td>
<td>Email service</td>
<td>Jun-06</td>
</tr>
<tr>
<td>QSpace/MySpace</td>
<td>Social networking</td>
<td>Nov-06</td>
</tr>
<tr>
<td>adultspace.com</td>
<td>Social networking</td>
<td>Dec-06</td>
</tr>
<tr>
<td>gaiaonline.com</td>
<td>Online gaming</td>
<td>Jan-07</td>
</tr>
<tr>
<td>u-dominion.com</td>
<td>Online gaming</td>
<td>Jan-07</td>
</tr>
</tbody>
</table>
Consequences?

- Samy took down MySpace (October 2005)
  - Site couldn’t cope: down for two days
  - Came down after 13 hours
  - Cleanup costs

- Yamanner (Yahoo mail) worm (June 2006)
  - Sent malicious HTML mail to users in the current user’s address book
  - Affected 200,000 users, emails used for spamming
Samy: Worm Propagation

- **Initial infection:**
  - Samy's MySpace page
  - Injected JavaScript payload exploits a XSS hole

- **Propagation step:**
  - User views an infected page
  - Payload executes:
    - Adds Samy as friend
    - Add payload to user's page
What’s at the Root of the Problem?

- Worms of the previous decade enabled by buffer overruns
- JavaScript worms are enabled by cross-site scripting (XSS)

- Fixing XSS holes is best, but some vulnerabilities remain
  - The month of MySpace bugs
  - Database of XSS vulnerabilities: xssed.com
Existing solutions rely on signatures
  - Ineffective: obfuscated and polymorphic JavaScript worms are very easy to write
  - Most real-life worms are obfuscated

Fundamental difficulties
  - **Server** can’t tell a user request from worm activity
  - **Browser** doesn’t know where JavaScript comes from
Spectator: Approach and Architecture
Worm Propagation

- $u_1$ uploads to his page
- $u_2$ downloads page of $u_1$
- $u_2$ uploads to his page
- $u_3$ downloads page of $u_2$
- $u_3$ uploads to his page
- ...

Propagation chain

1. Preserve causality of uploads, store as a graph
2. Detect long propagation chains
3. Report them as potential worm outbreaks
Spectator Architecture

Server-side application

page

Spectator proxy

page

Client-side tracking

request

U_1

U_2

request

tag
Causality Propagation on Client/Server

- Tagging of uploaded input

```html
<div spectator_tag=56>
  <b onclick="javascript:alert('…')">…</b>
</div>
```

- Client-side request tracking
  - Injected JavaScript and response headers
  - Propagates causality information through cookies on the client side
Propagation graph $G$:
- Records causality between tags (content uploads)
- Records IP address (approximation of user) with each

Worm: $Diameter(G) > \text{threshold } d$
Determining diameter precisely is exponential
Scalability is crucial
- Thousands of users
- Millions of uploads
Use greedy approximation of the diameter instead

<table>
<thead>
<tr>
<th></th>
<th>Precise algorithm</th>
<th>Approximate algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upload insertion time</td>
<td>$O(2^n)$</td>
<td>$O(1)$ on average</td>
</tr>
<tr>
<td>Upload insertion space</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Worm containment time</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Experiments
Experimental Overview

- Large-scale simulation with OurSpace:
  - Mimics a social networking site like MySpace
  - Experimented with various patterns of site access
  - Looked at the scalability

- Real-life case study:
  - Uses Siteframe, a third-party social networking app
  - Developed a JavaScript worm for it similar to real-life ones
Test-bed: OurSpace
- Every user has their own page
- At any point, a user can read or write to a page
  - `Write(U_1, “hello”); Write(U_1, Read(U_2)); Write(U_3, Read(U_1));`

Various access scenarios:
- **Scenario 1**: Worm outbreak (random topology)
- **Scenario 2**: A single long blog entry
- **Scenario 3**: A power law model of worm propagation
Tag addition overhead pretty much constant
Approximation of Graph Diameter

- Approximate worm detection works well

![Graph showing approximate and precise detection](chart.png)
Real-life worm experimentation is difficult.

- Used Siteframe, an open-source blogging system.
  - Found an exploitable XSS vulnerability.
  - Developed a worm for it.
- Scripted user behavior.
- Spectator flags the worm.
Conclusions

- First defense against JavaScript worms
  - Fast and slow, mono- and polymorphic worms
  - Scales well with low overhead

- Essence of the approach
  - Perform distributed data tainting
  - Look for long propagation chains

- Demonstrated scalability and effectiveness